## Trust Management and Security

This document gives an overview on the Hyperty Trust Model as well as on Hyperty Sandbox runtime execution environment.

Hyperties are securely associated to User Identities selected by the end-user himself. Hyperty Users are human beings (including group of human beings e.g. corporation) or things (including group of things and physical spaces e.g. a smart home or smart building).

Hyperty Trust Model extends [WebRTC Identity model](https://w3c.github.io/webrtc-pc/#sec.identity-proxy) where Identity tokens are generated, inserted in intercepted Messages sent by Hyperties and validated by recipient Hyperty Runtimes before delivered to the target Identity. These identity management procedures are performed according to applicable policies managed by the end-user.



Hyperty Trust Management

### User Identity

In our modern society, technology is ubiquitous, and transactions are evermore accomplished using digital technologies without the need to involve physical contact. An example of this situation can be observed in money transactions, whilst a few years ago if someone needed to make a bank transfer, it would require that person to move personally into a bank agency to order it, and in current days these money transfers can be performed using a smartphone. To achieve this, we need digital credentials to prove who we are and what we are allowed to do in remote communication. Therefore two important information security mechanisms must be implemented: authentication and authorization.

Authentication is verifying the identity claimed by an actor. Usually, to authenticate users, credentials make use of one or several factors among:

* something a user knows (such as a PIN code or a password),
* something a user owns (such as a SIM card),
* something a user is (such as a fingerprint, or a voice sample),
* something a user does (such as typing characteristics).

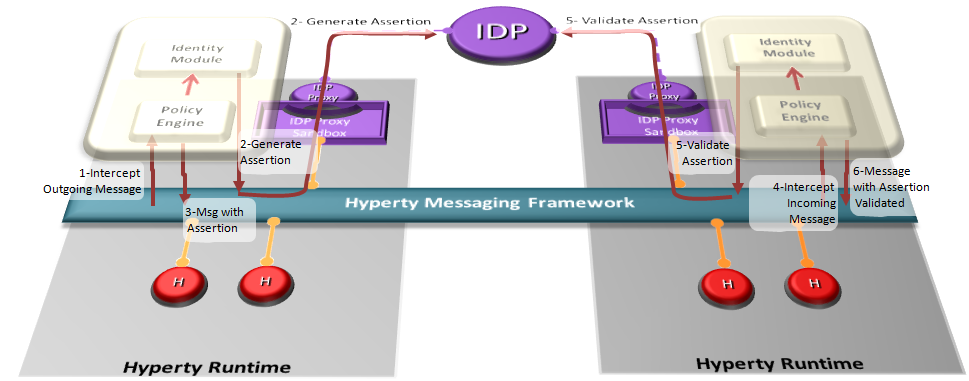
Authorization is deciding whether a given identity may execute or access a certain resource. Access control to a service or system, can be achieved based on access rights or policies that allow or deny a particular action based on an identifier, a role (RBAC), or an attribute (ABAC).

Hyperty trust model relies on service-independent authentication that is global and non-service-bound. In implementations, each Hyperty service provider may include their own user recognition (i.e. their own internal user accounts) and service authorization (i.e. level of permissions to use the service), over and above the initial user identification.

### Identity Module and IdP Proxy

The Identity Module (Id Module) is the [Core Runtime](https://github.com/reTHINK-project/dev-runtime-core) component responsible for handling the user identity and the association of this identity with the Hyperty instances. The identity in the reTHINK project is not fixed to a unique Identity source (espically it is not bound to a single Service Provider), but obtained through several Identity P£rovider. In this approach, the Id Module provides the user the option to choose the preferred IdP for authentication. For example, a user with a Google account can use Google as an Identity Provider to retrieve Identity Tokens that may be associated by the Identity Module with a Hyperty instance. Following the flexible and adaptable approach fostered by WebRTC standards (from both [IETF](https://tools.ietf.org/html/draft-ietf-rtcweb-security-arch-12#section-5.6.2) and [W3C](https://www.w3.org/TR/webrtc/#sec.identity-proxy)) IdP Proxies are downloaded from each Identity Provider to act as intermediaries between the Identity Module and the aforementioned Identity Providers. The IdP Proxy is the [IdP ProtoStub](hyperty-messaging-framework.md#protocol-on-the-fly-protofly-and-protostubs) that is responsible to handle the communication between the Identity Module and a specific Identity Provider.

The following figure illustrates this interaction:



Interaction between the Identity Module and the Identity Provider

*NB. The Policy Engine is the Core Runtime component rsponsible to manage and apply policies to perform access control. More details are found in the* [*Policy Management*](https://github.com/reTHINK-project/specs/tree/master/policy-management) *section.*

### Mutual authentication

Identity management relies upon managing life-cycle of identity-related security tokens. Therefore, identity assertions are used in reTHINK to identify a user and to prove that he was authenticated on an IdP. In addition, whenever a user intends to initiate a communication with another user, the mutual authentication protocol is triggered so that users can authenticate each other mutually.

The full specification of reTHINK Mutual authentication is provided [here](mutual-authentication.md)

### Secure Communication

The communication between users is one of the major characteristic in reTHINK. As such, the authentication of each user takes a big role to ensure that no personification can occur. Every time a user starts a communication with another user, the process of mutual authentication, described above, is initiated by the Identity Module. This mutual authentication is not only useful for the authentication of the users, but is also essential for the exchange of the symmetric keys used in the established secure communications. In case, one of the users starts a communication anonymously, the authentication of the other user is made, so it can be possible to establish a secure channel between the two users.

Further details are provided [here](secure-communication.md).

### Identity selection

The Identity Module is responsible to obtain, manage the identities in reTHINK, and provide the identities when the Runtime Registry requires an identity to associate with a Hyperty. For the users to be able to select the desired identity, a GUI is mandatory. An identity GUI is part of the reTHINK web browser application, and allows the user the act of registering a Hyperty to select the identity he wants to associate with it.

The identity GUI can be presented in two distinct ways: by clicking in the settings button on the reTHINK web application, or by clicking in the button to register a Hyperty that will trigger the appearance of the identity GUI. This GUI allows the user to add new identities from a list of IdPs, choose an identity from the existing ones and remove identities from the list of previously registered identities. For example, when a user using the reTHINK web application click the button to register a new Hyperty the identity GUI appears listing all identities previously registered, and the option to add new ones. After the user selects the desired identity or chooses to obtain a new one, the Hyperty registration process continues.

### Runtime Sandbox

The hyperty runtime implements sandboxing mechanisms that ensure the correct isolation of client JavaScript code (i.e., Hyperties, ProtoStubs, and Applications). Isolation means that client code is confined to execute within the address space of an independent sandbox. As a result, sandboxes prevent potentially malicious code from interfering with client code instances in co-located sandboxes or from accessing external resources in the surrounding environment (e.g., files, network, etc.). Communication outside the sandbox is possible through well defined channels. Both sandboxing mechanisms and communication channels implemented by the hyperty runtime are available to the application programmer throught specific APIs and are dependent on the targeted platform.

For the browser platform, sandboxing is enforced by leveraging native mechanisms provided by the browser API. The core runtime components execute inside an iFrame. The iFrame implements the core sandbox by isolating the code of the core runtime from the main window in which the application Javascript code is executed. Application code is therefore prevented from accessing directly to the memory address space of the core runtime. Communication between application and core runtime is possible only through a single and well defined entrypoint which allows them to exchange messages: method postMessage(). Hyperties and protoStub downloaded from different domains are executed in independent Web Workers. Web Workers effectively isolate their internal states from each other and from the core runtime. The postMessage() method constitutes the only communication bridge between the these components.

For the standalone platform, the sandboxing mechanisms we employ are similar to the browser platform. The main difference is that, instead of using a browser, we leverage Crosswalk to support standalone applications. Crosswalk is an HTML application runtime that allows us to execute the hyperty runtime as native mobile applications in Android and iOS devices without the need to install a full-blown browser. Mobile applications only need to be bundled with both Crosswalk webviews and the hyperty runtime code. Since a Crosswalk webview implements a Chromium-based runtime, it can seamlessly execute the hyperty runtime code that was implemented for the browser platform.